# Analyzing EfficientNet on grading Diabetic Retinopathy

*Ayush Shirsat, Sarthak Arora*

ayush34@bu.edu, sarthak@bu.edu

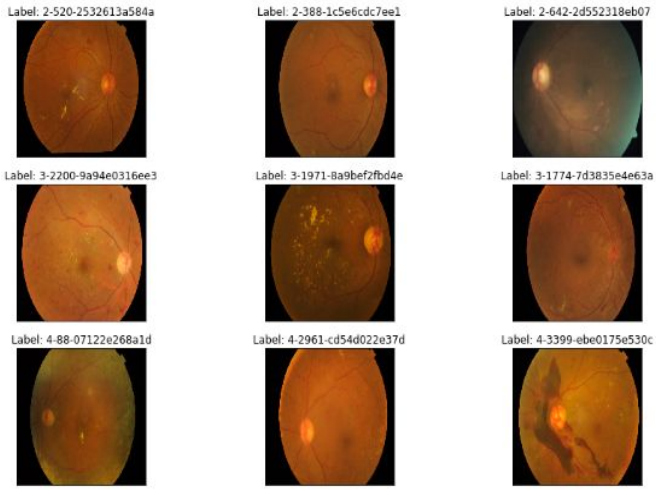


Figure 1. Retina images

[Source:https://www.kaggle.com/ratthachat/aptos-eye-preprocessing-in-diabetic-retinopathy]

## 1. Task

Aim of this project is to perform model scaling and balance for Convolutional Neural Networks (ConvNets) through the performance of EfficientNet models on the diabetic retinopathy dataset. We grade the severity of Diabetic Retinopathy (DR) using Deep learning models. Retina images as shown in Figure.1 are captured using a fundus camera. These images are under different lighting conditions and are not necessarily centered which pose a challenge that we solve using image pre-processing. Diabetic retinopathy is generally graded on a scale of 0-4, with 0 being no DR and 4 being proliferative DR. We analyzed the performance of EfficientNets on this dataset by scaling depth, width and resolution.

## 2. Related Work

Our primary reference paper is related to EfficientNet [1]. This paper talks about how a new uniform scaling method that balances the network depth, width, and resolution leads to a better performance in ConvNets. The family of models, called EfficientNets have been shown to achieve better accuracy and efficiency than other ConvNets like ResNet, MobileNets. The scaling proposed by these authors is called compound scaling and it results in defining scaling coefficients. They use a grid search method to find the relationship between different scaling methods on the baseline model. EfficientNets use Swish as their activation function [2]. The paper observes that different scaling dimensions like network depth, network width, and image resolution are not independent. Different scaling dimensions are coordinated and balanced rather than single-dimension scaling. A compound scaling method was proposed and led EfficientNet-B7 to achieve 84.4% top-1 and 97.1% top-5 accuracy on ImageNet, while being 8.4x smaller and 6.1x faster on inference than the previous best ConvNet as shown in Figure 2.

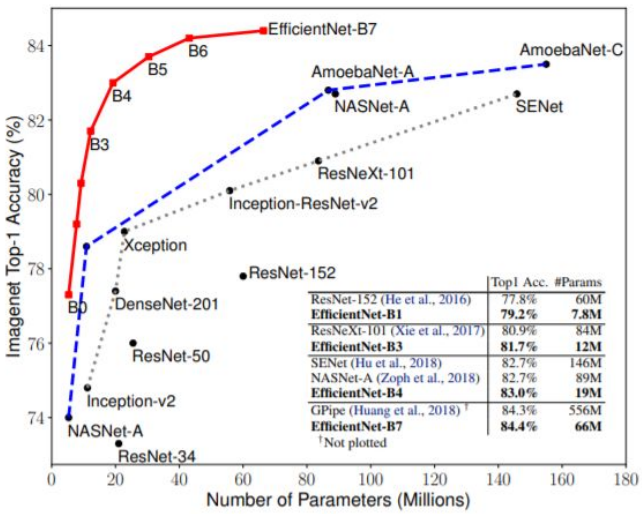


Figure 2. Comparing EfficientNet with other ConvNets

As we go on adding more layers the model starts to become less efficient due to the effect of vanishing gradients. To mitigate this issue a skip architecture is proposed in paper [3]. This allowed models to go deeper and resulted in ResNet-34, ResNet-50, ResNet-150, etc. This paper also demonstrated that just going deeper does not help, like ResNet-1000 is not much accurate to resnet-152. This basically led to saturation of accuracies and they did not improve. The

models also need to go wider, or have more number of channels to improve accuracy further. Hence, scaling width becomes important.

The compound scaling method is as follows:

$$
\begin{aligned}
depth: \ d &= \alpha^{\varphi} \\
width: \ w &= \beta^{\varphi} \\
resolution: \ r &= \gamma^{\varphi} \\
s.t. \ \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\
\alpha \geq 1, \ \beta &\geq 1, \ \gamma \geq 1
\end{aligned}
$$

α, β, γ were obtained from grid search method and then fixed based on the B0 model. Φ is a user defined coefficient that mentions the resources available for model scaling and is varied to obtain B1-B7 models.

One of the modules in EfficientNet is the MBConv block, which was first introduced as a mobile architecture in the form of MobileNetV2 [4]. This block comprises a module known as the inverted residual with linear bottleneck as shown in Figure 3. Layers with a large number of channels are connected in normal residual blocks whereas, here, the bottlenecks are connected. With this module, the bottleneck layers are mapped to a higher dimension and the corresponding features are mapped back to lower dimension using linear convolution. In normal residual networks we generally map feature maps as wide-narrow-wide architecture but in this case it is a narrow-wide-narrow architecture. Hence, the name “inverted bottleneck”. This sort of architecture reduces the model parameters significantly which helps us to create deeper models with consistent improvements in accuracies.

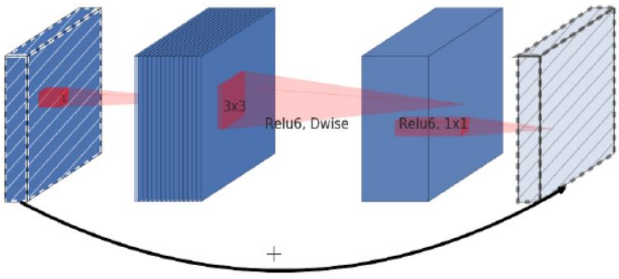


Figure 3. MBConv block

[Source:https://towardsdatascience.com/mobilenetv2-inverted-residuals-and-linear-bottlenecks-8a4362f4ffd5]

The EfficientNet architecture is explained as follows. The input image first passes through a Squeeze-Excitation (SE) network (narrow) as shown in Figure 4. The outputs are now passed through a depthwise convolution layer which scales up the number of channels (wide). The outputs (X) of this depthwise convolution is now followed with another SE network (narrow). For the working of the SE network, we can assume the Residual block to be another convolution layer. First each channel is squeezed to a single numeric value using Global Average pooling. It then passes through a layer of Convolution which is the first fully connected (FC) layer where it is downsampled and then passed again through a convolution layer which is a second FC layer, where it is upsampled to its original channel length. This output is multiplied with the output of our Residual block. Basically it now acts as weights for our original Residual feature maps with each channel now scaled. Finally we add X to this scaled output using skip connection and our MBConv block is now ready. Several such MBConv blocks are used to make a family of EfficientNets.

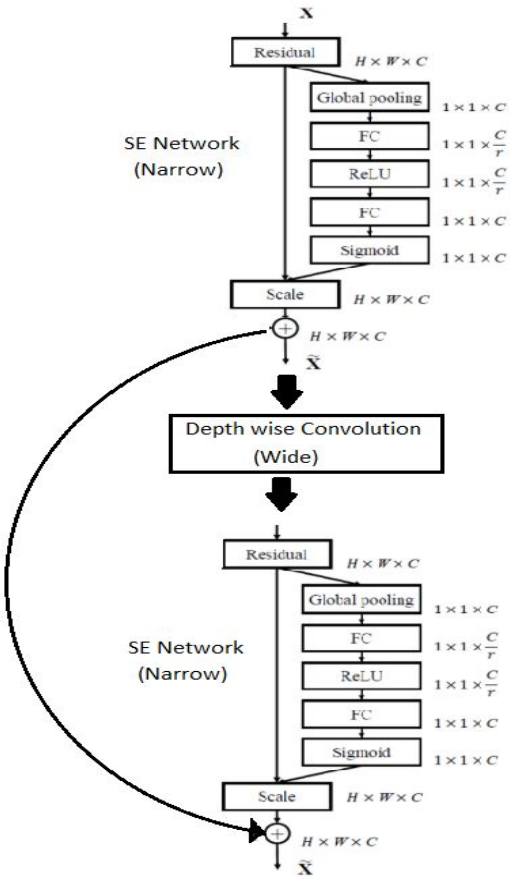


Figure 4. Squeeze and Excitation Network

ReLU is the most widely used activation function however, Swish outperforms it. On CIFAR-100 ReLU

had accuracy of 74.2% and Swish had an accuracy of 75.1% on a ResNet model. Swish again outperformed ReLU on ImageNet dataset using MobileNet with 74.2% accuracy, while ReLU had only 72% accuracy.

Autograding of retina images using deep learning methods achieve accuracy equal to or better than manual grading by ophthalmologists as mentioned in paper [6]. Autograding is also an efficient approach to diagnosing the patients as manual grading requires time and specialized training.

**3. Approach**

Our dataset is available via a Kaggle competition and we made an inference kernel which is separate from our training kernel to test our model performance on the Kaggle test set. We are using SCC, Kaggle and Google colab environments to run our kernels. TensorFlow and Keras are used as our primary frameworks.

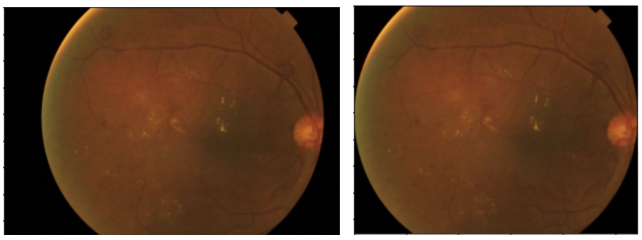

Figure 5. Original image(above) is cropped and center adjusted for preprocessing

For preprocessing the images were cropped and center adjusted as shown in Figure 5 with the help of a mask. This mask was generated based on the fact that there were several black pixels and if the pixel value was below a certain threshold or tolerance the mask would be False at that image index.

Some of the augmentations that we implemented on the data were smoothing the images by Median or Gaussian filtering, flipping the images horizontally vertically, or combining the two if needed. These augmentations are shown in Figure 6. Rotations over different angles were also tried, however, the augmented images after rotation were leading to either loss of information in the image or adding unnecessary details so we decided to remove them. Our total dataset consisted of 38,789 samples.

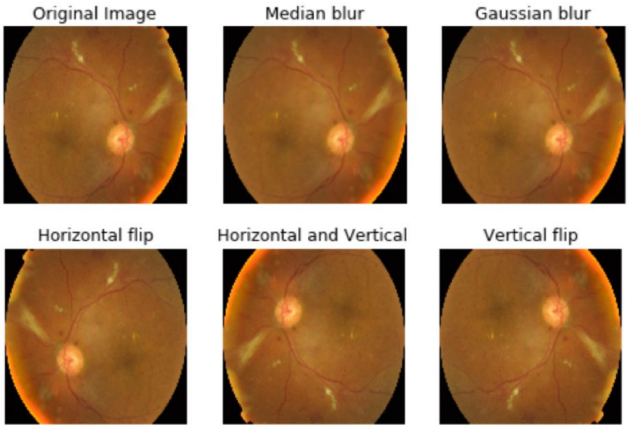


Figure 6. Data Augmentations

We trained and tested models from B0 to B7. Transfer learning was performed using pre-trained ImageNet weights for EfficientNet.

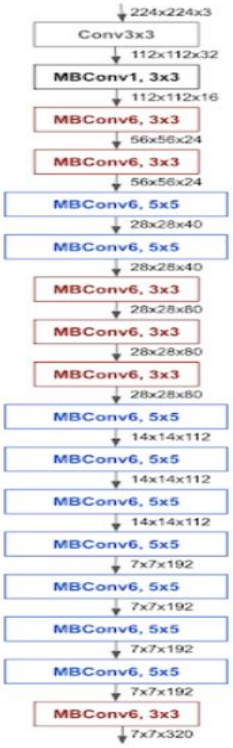


Figure 7. EfficientNet-B0
[Source:https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html]

EfficientNets are a family of models that are scaled up from the baseline model called EfficientNet-B0 shown in Figure 7. We perform compound scaling in which the

width, depth and resolution is scaled by a fixed set of scaling coefficients.

Coefficients are as follows:

- Depth (No. of layers): 1.20
- Width (No. of channels): 1.10
- Resolution (Image size): 1.15

The architecture of the B0 model starts with a convolution layer followed by layers of mobile inverted bottleneck convolution layers

Swish activation function is used in implementation of EfficientNets given by formula:

$$f(x) = x \cdot sigmoid(\beta x)$$

Few layers are added on top of the efficientNet model and the full architecture is shown in Figure 8. The purpose of these layers is just to map outputs of EfficientNet models and avoid overfitting by using dropout and batch normalization layers

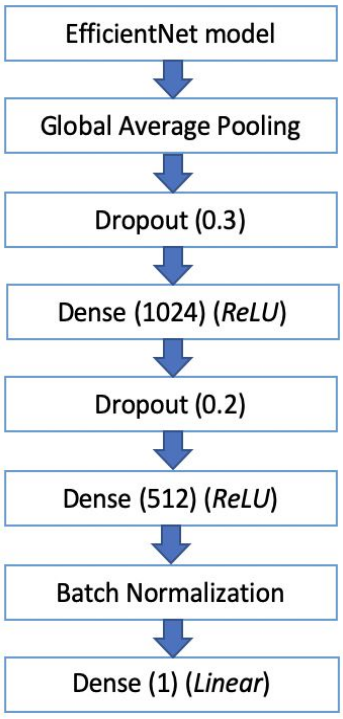


Figure 8. Full Architecture

Table 1. Shows the number of parameters and number of layers for each model. We plot the test scores with respect to the Number of Parameters, which helped us analyze EfficientNets performance.

| Model | Number of Parameters | Number of layers | Image Resolution |
|---|---|---|---|
| B0 | 5.89 M | 237 | 224 |
| B1 | 8.42 M | 339 | 240 |
| B2 | 9.74 M | 339 | 280 |
| B3 | 12.89 M | 384 | 300 |
| B4 | 20.04 M | 474 | 380 |
| B5 | 31.14 M | 576 | 456 |
| B6 | 43.85 M | 666 | 528 |
| B7 | 67.25 M | 813 | 600 |

Table 1. B0-B7 models

We had used loss function as categorical cross-entropy but decided to switch to Mean Squared Error (MSE) to perform regression. We also added a custom metric called Quadratic Weighted Kappa (QWK) as it was used in Kaggle as a score metric. The QWK is a callback which is called after every epoch and it prints the QWK score based on validation data. We tried to maximize our QWK score and saved the best weights on the highest score. We changed our model from classification to regression and did see improvements in score.

Regression generally gives an estimate of values which are float but we need integer values to grade the severity. For this we created a Rounder class which would map our regression values to Grade score. It was done by picking 4 coefficients as follows:

[coef0,coef1, coef2 coef3 ] = [0.5, 1.5, 2.5, 3.5]

If the predicted value was less than 0.5 the model would predict 0. If value is between 0.5 and 1.5 prediction would be 1, if value is between 1.5 and 2.5 prediction is 2, if value is between 2.5 and 3.5 prediction is 3 else prediction is 4.

Quadratic Weighted Kappa or QWK is the evaluation metric for the Kaggle competition. The data is rated by humans and the submission predictions are mapped with the human diagnosis. The QWK metric rates between 0 (random agreement between raters) and 1 (complete agreement between raters). If the agreement between the human diagnosis and

submitted prediction is less than that given by a random guesser, the metric may go below 0.

Dataset is split in train and validation sets (80%-20% split). Hyper-parameters are tuned based on validation loss. Learning rate scheduler is implemented to fine tune our model and the rates are specific to the EfficientNet model. Best weights were saved on kappa metric and val loss. Most models were trained to 24 epochs but last two were pushed to 30 epochs. Models could perform an early stop if validation loss started to increase. This is done to prevent overfitting.

We are also plotting the values of train and validation loss. This helped us identify overfitting or the epoch at which the model gave least validation loss and best kappa score and hence, best weights.

**4. Dataset and Metric**

There are two datasets that are available on Kaggle from 2 different competitions. Both datasets have massive class imbalance. We have introduced augmentations, as shown in Figure 9, to even things out.

Dataset repositories:

- https://www.kaggle.com/c/aptos2019-blindness-detection/data (3662 training samples)
- https://www.kaggle.com/c/diabetic-retinopathy-detection/data (35127 training samples)

Since we used kaggle competition to evaluate on test data we had 1928 test examples that are public and approximately 13000 test examples that are private. The private test samples can only be evaluated after submitting an inference kernel to Kaggle. The distribution of training data is shown in Figure 9.

Pre-trained ImageNet weights:

- https://www.kaggle.com/ratthachat/efficientnet-keras-weights-b0b5/

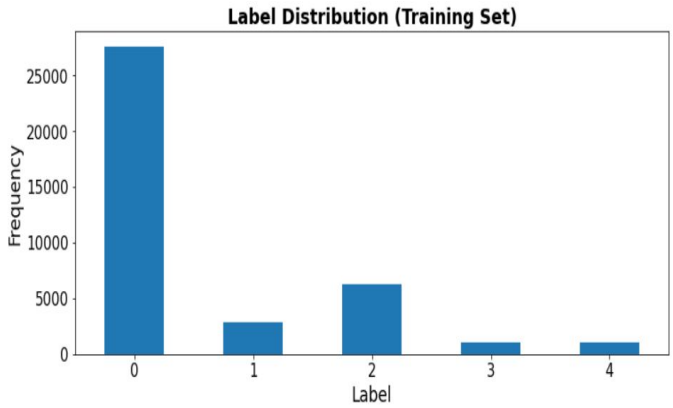


Figure 9. Label distribution

We evaluated the train and validation loss and kappa metric. As a score metric quadratic weighted kappa was computed as per competition guidelines (https://www.kaggle.com/c/aptos2019-blindness-detection/overview/evaluation). Initially, our loss function was categorical cross entropy as we were performing classification but we shifted to regression and changed the loss to Mean Squared Error (MSE). MSE is quite similar to Quadratic weighted Kappa score and hence, is able to optimize well.

EfficientNet repository:

- https://github.com/qubvel/efficientnet

In the process of evaluating EfficientNets we compared the test scores with respect to number of model parameters (as we varied from B0 to B7). We also mention the training time it took for all models.

Quadratic Weighted Kappa measures the rating agreement between the two raters. Here, the raters are, the given human readings and our model predictions. The metric penalizes by taking the difference between the ratings and squaring them. The range for qwk is -1 (farthest possible disagreement between ratings) to 1 (complete agreement between ratings).

$$w_{i,j} = \frac{(i-j)^2}{(N-1)^2}$$

$$\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}}$$

$w_{i,j}$ forms the weight matrix that punishes the ratings quadratically. N indicates the number of classes. In our case N is 5. $i$ is the actual rating (true label) as given by a human and $j$ is the predicted rating. O is a confusion matrix over the actual and predicted ratings and E is the expected matrix. E is calculated with the assumption that there's no correlation between the two ratings and is formed by the outer product between the value counts of actual and predicted vectors. The outer product between two N length vectors yields a NxN matrix. Both O and E are normalized.

**5. Results**

Figure 10. Shows the distribution of our predicted labels on a public test set. Our scores are based on a private test set which is not open to the public. This is done by Kaggle to avoid cheating by participants. Our inference kernel is submitted to kaggle which performs predictions on a private test set and kaggle evaluates our score. The distribution of the public test set is not similar to our training data which is alright and it was intentional by the competition organizers. The private set which comprises public test data and additional test data has a similar distribution as the training set. This data is not available for us to see but our QWK score on our validation set almost matched our QWK score on the private test set.

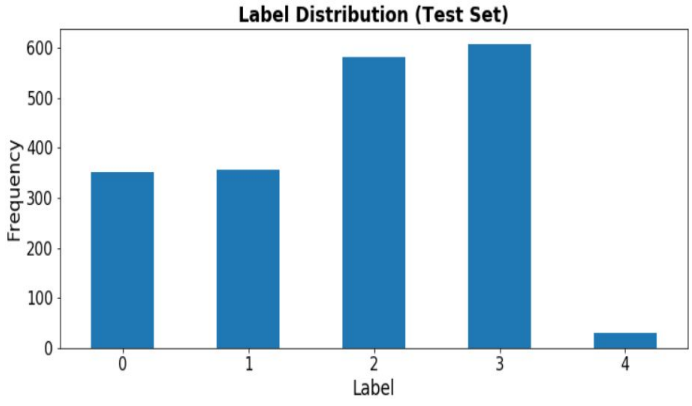


Figure 10. Label Distribution (Public Test Set)

| Model | Epochs | Batch Size | Training Time | Test Score |
|---|---|---|---|---|
| B0 | 23 | 16 | 3.5 hrs | 0.715 |
| B1 | 25 | 16 | 4.5 hrs | 0.740 |
| B2 | 25 | 16 | 6 hrs | 0.801 |
| B3 | 25 | 16 | 10 hrs | 0.823 |
| B4 | 25 | 12 | 16 hrs | 0.834 |
| B5 | 25 | 8 | 21 hrs | 0.840 |
| B6 | 30 | 4 | 26 hrs | 0.849 |
| B7 | 30 | 4 | 30 hrs | 0.856 |

Table 2. B0-B7 model results

We trained and tested B0-B7 models and our test scores seem to increase as we scaled up the models. In Table 2, we have mentioned the number of epochs it took to train these respective models. In some models like B0, the early stopping callback, due to the validation loss not improving, led to the number of epochs being less. Along with the number of epochs, we have mentioned the training time it took for the corresponding model.

We have also shown the batch size as it may have affected training, predictions and time taken. We had initially planned to keep batch size of 16 constant for all models. But we soon faced issues with RAM and our model did not start training. To mitigate this issue we reduced batch size as we went on scaling our model.

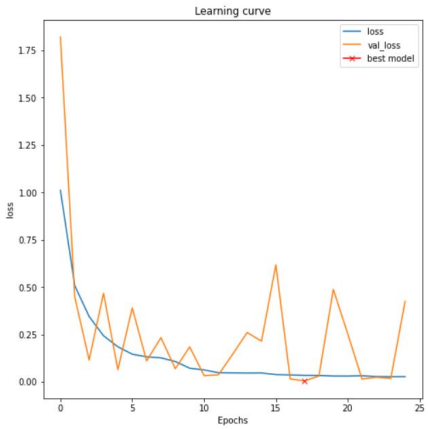


Figure 11. Learning curve and Training curve for B3

For the models, we are plotting loss and validation loss over the course of epochs as presented in the Learning Curve. In Figure 11., we have shown this curve for EfficientNet B3. For other models we had some restrictions with session times so we had to save weights and retrain again in a different session. We could not generate the learning curves for models B4 and above.

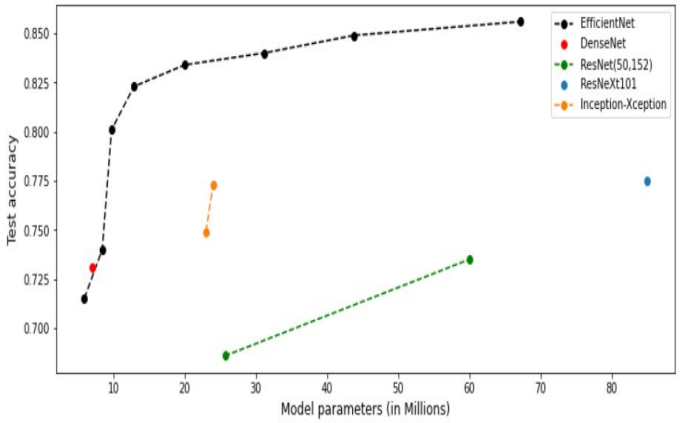


Fig 12. Comparing EfficientNet with other models

Figure 12 shows the performance of EfficientNets against other models. We compared the public kernels and their scores and plotted it with our readings. As expected we can see EfficientNets outperformed all these models and gave better test scores with less model parameters.

## 6. Conclusion

We observed Diabetic Retinopathy images through B0-B7 models and observed as the models scaled up the qwk scores improved. The training time and model parameters also climb up as the model grows. However, the compound scaling is efficient for convolutional neural networks as it provides better results and takes less training time than single dimension scaled ConvNets. The number of model parameters is also fewer and thus efficient as compared to other ConvNets. All this resulted in the family of EfficientNet models to be faster on inference as well. EfficientNets also outperformed all other single models in the competition.

We wanted to maintain a constant batch size of 16 but were not able to do so because of Memory issues with RAM. Reducing batch size to 4 may or may not have affected our test scores. We still were able to replicate a similar curve for test accuracy and model parameters as the original EfficientNet paper.

## 7. Timeline

| Task | Deadline | Lead |
|---|---|---|
| Data cleaning/ Pre-processing | 03/11/20 | Sarthak |
| Prepare inference kernel | 03/11/20 | Ayush |
| Train and finetune B0 and B2 models | 04/01/20 | Ayush |
| Train and finetune B1 and B3 model | 04/01/20 | Sarthak |
| Prepare update report | 04/06/20 | All |
| Train and finetune B4 and B6 models | 04/24/20 | Ayush |
| Train and finetune B5 and B7 model | 04/24/20 | Sarthak |
| Presentation | 04/27/20 | All |
| Report | 05/01/20 | All |

Table 3. Project timeline

**Github Link**

https://github.com/Ayush-Shirsat/EfficientNet_DR

**References**


1) Tan, Mingxing, and Quoc V. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." arXiv preprint arXiv:1905.11946 (2019).

2) Ramachandran, Prajit, Barret Zoph, and Quoc V. Le. "Searching for activation functions." arXiv preprint arXiv:1710.05941 (2017).

3) He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

4) Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

5) Hu, Jie, Li Shen, and Gang Sun. "Squeeze-and-excitation networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

6) Gulshan, Varun, et al. "Performance of a deep-learning algorithm vs manual grading for detecting diabetic retinopathy in india." JAMA ophthalmology 137.9 (2019): 987-993.